本篇将分三种不同的情况下讨论springMvc的容器的创建过程

情况1:只配置了DispatcherServlet

web.xml的配置

```
<servlet>
    <servlet-name >spring-mvc</servlet-name>
    <!-- servlet类 -->
    <servlet-class >org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!-- 初始化参数 -->
    <init-param >
        <param-name >contextConfigLocation</param-name>
        <param-value >classpath:/spring-mvc.xml</param-value>
    </init-param>
    <!-- 启动时加载 -->
    <load-on-startup >1</load-on-startup>
</servlet>
<servlet-mapping >
    <servlet-name >spring-mvc</servlet-name>
    <url-pattern >/</url-pattern>
</servlet-mapping>
```

这种情况下只是在容器(服务器Tomcat或者Jetty都算)启动时创建一个ApplicationContext容器 基本流程为:

1.创建并刷新容器

```
//FrameworkServlet.java文件
@Override
protected final void initServletBean() throws ServletException {
    getServletContext().log("Initializing Spring FrameworkServlet '" + getServletName() + "'");
    if (this.logger.isInfoEnabled()) {
        this.logger.info("FrameworkServlet '" + getServletName() + "': initialization started");
    }
    long startTime = System.currentTimeMillis();

    try {
        //初始化ApplicationContext,能从环境中获取就获取没有就创建
        this.webApplicationContext = initWebApplicationContext();
        initFrameworkServlet();
    }
    catch (ServletException ex) {
        this.logger.error("Context initialization failed", ex);
        throw ex;
    }
    catch (RuntimeException ex) {
        this.logger.error("Context initialization failed", ex);
        throw ex;
    }

    if (this.logger.isInfoEnabled()) {
        long elapsedTime = System.currentTimeMillis() - startTime;
```

```
        this.logger.info("FrameworkServlet '" + getServletName() + "': initialization completed in
" +
                elapsedTime + " ms");
    }
}

protected WebApplicationContext initWebApplicationContext() {
        //在环境中获取ApplicationContext容器 此时获取不到
        WebApplicationContext rootContext =
                WebApplicationContextUtils.getWebApplicationContext(getServletContext());
        WebApplicationContext wac = null;

        if (this.webApplicationContext != null) {
            // A context instance was injected at construction time -> use it
            wac = this.webApplicationContext;
            if (wac instanceof ConfigurableWebApplicationContext) {
                ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) wac;
                if (!cwac.isActive()) {
                    // The context has not yet been refreshed -> provide services such as
                    // setting the parent context, setting the application context id, etc
                    if (cwac.getParent() == null) {
                        // The context instance was injected without an explicit parent -> set
                        // the root application context (if any; may be null) as the parent
                        cwac.setParent(rootContext);
                    }
                    configureAndRefreshWebApplicationContext(cwac);
                }
            }
        }
        if (wac == null) {
            // No context instance was injected at construction time -> see if one
            // has been registered in the servlet context. If one exists, it is assumed
            // that the parent context (if any) has already been set and that the
            // user has performed any initialization such as setting the context id
            wac = findWebApplicationContext();
        }
        if (wac == null) {
            // No context instance is defined for this servlet -> create a local one
            //创建容器并初始化和刷新容器(刷新之前有详细介绍)
            wac = createWebApplicationContext(rootContext);
        }

        if (!this.refreshEventReceived) {
            // Either the context is not a ConfigurableApplicationContext with refresh
            // support or the context injected at construction time had already been
            // refreshed -> trigger initial onRefresh manually here.
            onRefresh(wac);
        }

        if (this.publishContext) {
            // Publish the context as a servlet context attribute.
            String attrName = getServletContextAttributeName();
            getServletContext().setAttribute(attrName, wac);
            if (this.logger.isDebugEnabled()) {
                this.logger.debug("Published WebApplicationContext of servlet '" +
getServletName() +
                        "' as ServletContext attribute with name [" + attrName + "]");
```

```
            }
        }

        return wac;
    }
```

2.刷新完毕后触发事件初始化springMvc9大组件

```
protected void finishRefresh() {
    // Initialize lifecycle processor for this context.
    initLifecycleProcessor();

    // Propagate refresh to lifecycle processor first.
    getLifecycleProcessor().onRefresh();

    // Publish the final event.
    //发布IOC容器刷新完毕事件
    publishEvent(new ContextRefreshedEvent(this));

    // Participate in LiveBeansView MBean, if active.
    LiveBeansView.registerApplicationContext(this);
}
//收到事件调用onRefresh()方法初始化springMvc的9大组件
public void onApplicationEvent(ContextRefreshedEvent event) {
    this.refreshEventReceived = true;
    onRefresh(event.getApplicationContext());
}
protected void onRefresh(ApplicationContext context) {
    //初始化9大组件
    initStrategies(context);
}
//9大组件初始化
protected void initStrategies(ApplicationContext context) {
    initMultipartResolver(context);
    initLocaleResolver(context);
    initThemeResolver(context);
    initHandlerMappings(context);
    initHandlerAdapters(context);
    initHandlerExceptionResolvers(context);
    initRequestToViewNameTranslator(context);
    initViewResolvers(context);
    initFlashMapManager(context);
}
```

情况2:同时配置了DispatcherServlet与ContextLoaderListener的情况

```
<servlet>
    <servlet-name >spring-mvc</servlet-name>
    <!-- servlet类 -->
    <servlet-class >org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!-- 初始化参数 -->
    <init-param >
        <param-name >contextConfigLocation</param-name>
        <param-value >classpath:/spring-mvc.xml</param-value>
    </init-param>
```

```
    <!-- 启动时加载 -->
    <load-on-startup >1</load-on-startup>
</servlet>
<servlet-mapping >
    <servlet-name >spring-mvc</servlet-name>
    <url-pattern >/</url-pattern>
</servlet-mapping>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

这种情况下,容器(服务器)启动后,ContextLoaderListener监听到容器启动后,直接创建一个IOC容器并执行刷新操作，然后再到DispatcherServlet的情况，将以先前所创的IOC容器为父容器，再创建一个子IOC容器，并刷新子容器，最后再监听到子容器刷新完毕，初始化springMvc9大组件。

基本流程:

1.容器(服务器)启动后,ContextLoaderListener监听到容器启动后,直接创建一个IOC容器并执行刷新操作

```
//ContextLoaderListener.java文件
@Override
public void contextInitialized(ServletContextEvent event) {
    //监听到服务器启动,初始化IO容器
    initWebApplicationContext(event.getServletContext());
}
public WebApplicationContext initWebApplicationContext(ServletContext servletContext) {
    if (servletContext.getAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE)
!= null) {
        throw new IllegalStateException(
                "Cannot initialize context because there is already a root application context
present - " +
                "check whether you have multiple ContextLoader* definitions in your web.xml!");
    }

    Log logger = LogFactory.getLog(ContextLoader.class);
    servletContext.log("Initializing Spring root WebApplicationContext");
    if (logger.isInfoEnabled()) {
        logger.info("Root WebApplicationContext: initialization started");
    }
    long startTime = System.currentTimeMillis();

    try {
        // Store context in local instance variable, to guarantee that
        // it is available on ServletContext shutdown.
        if (this.context == null) {
            //创建容器
            this.context = createWebApplicationContext(servletContext);
        }
        if (this.context instanceof ConfigurableWebApplicationContext) {
            ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext)
this.context;
            if (!cwac.isActive()) {
                // The context has not yet been refreshed -> provide services such as
                // setting the parent context, setting the application context id, etc
                if (cwac.getParent() == null) {
```

```
                    // The context instance was injected without an explicit parent ->
                    // determine parent for root web application context, if any.
                    ApplicationContext parent = loadParentContext(servletContext);
                    cwac.setParent(parent);
                }
                //配置并刷新
                configureAndRefreshWebApplicationContext(cwac, servletContext);
            }
        }
        servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE,
this.context);

        ClassLoader ccl = Thread.currentThread().getContextClassLoader();
        if (ccl == ContextLoader.class.getClassLoader()) {
            currentContext = this.context;
        }
        else if (ccl != null) {
            currentContextPerThread.put(ccl, this.context);
        }

        if (logger.isDebugEnabled()) {
            logger.debug("Published root WebApplicationContext as ServletContext attribute with
name [" +
                    WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE + "]");
        }
        if (logger.isInfoEnabled()) {
            long elapsedTime = System.currentTimeMillis() - startTime;
            logger.info("Root WebApplicationContext: initialization completed in " + elapsedTime +
" ms");
        }

        return this.context;
    }
    catch (RuntimeException ex) {
        logger.error("Context initialization failed", ex);
        servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE,
ex);
        throw ex;
    }
    catch (Error err) {
        logger.error("Context initialization failed", err);
        servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE,
err);
        throw err;
    }
}
```

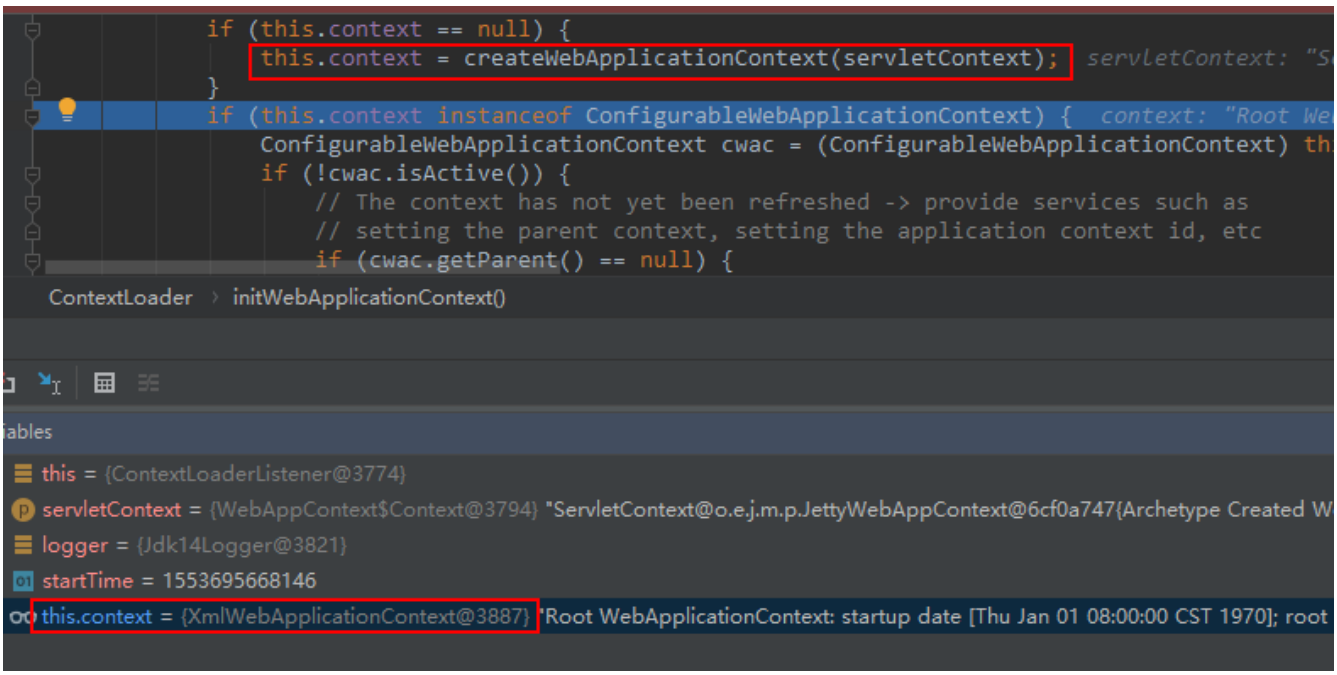

图1:父容器

2.子容器的创建刷新并初始化springMvc的9大组件

```
FrameworkServlet.java文件
@Override
protected final void initServletBean() throws ServletException {
    getServletContext().log("Initializing Spring FrameworkServlet '" + getServletName() + "'");
    if (this.logger.isInfoEnabled()) {
        this.logger.info("FrameworkServlet '" + getServletName() + "': initialization started");
    }
    long startTime = System.currentTimeMillis();

    try {
        this.webApplicationContext = initWebApplicationContext();
        initFrameworkServlet();
    }
    catch (ServletException ex) {
        this.logger.error("Context initialization failed", ex);
        throw ex;
    }
    catch (RuntimeException ex) {
        this.logger.error("Context initialization failed", ex);
        throw ex;
    }

    if (this.logger.isInfoEnabled()) {
        long elapsedTime = System.currentTimeMillis() - startTime;
        this.logger.info("FrameworkServlet '" + getServletName() + "': initialization completed in
" +
                elapsedTime + " ms");
    }
}
protected WebApplicationContext initWebApplicationContext() {
    //获取到父容器
    WebApplicationContext rootContext =
```

```
            WebApplicationContextUtils.getWebApplicationContext(getServletContext());
    WebApplicationContext wac = null;

    if (this.webApplicationContext != null) {
        // A context instance was injected at construction time -> use it
        wac = this.webApplicationContext;
        if (wac instanceof ConfigurableWebApplicationContext) {
            ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) wac;
            if (!cwac.isActive()) {
                // The context has not yet been refreshed -> provide services such as
                // setting the parent context, setting the application context id, etc
                if (cwac.getParent() == null) {
                    // The context instance was injected without an explicit parent -> set
                    // the root application context (if any; may be null) as the parent
                    cwac.setParent(rootContext);
                }
                configureAndRefreshWebApplicationContext(cwac);
            }
        }
    }
    if (wac == null) {
        // No context instance was injected at construction time -> see if one
        // has been registered in the servlet context. If one exists, it is assumed
        // that the parent context (if any) has already been set and that the
        // user has performed any initialization such as setting the context id
        wac = findWebApplicationContext();
    }
    if (wac == null) {
        // No context instance is defined for this servlet -> create a local one
        //传入父容器再创建子容器
        wac = createWebApplicationContext(rootContext);
    }

    if (!this.refreshEventReceived) {
        // Either the context is not a ConfigurableApplicationContext with refresh
        // support or the context injected at construction time had already been
        // refreshed -> trigger initial onRefresh manually here.
        onRefresh(wac);
    }

    if (this.publishContext) {
        // Publish the context as a servlet context attribute.
        String attrName = getServletContextAttributeName();
        getServletContext().setAttribute(attrName, wac);
        if (this.logger.isDebugEnabled()) {
            this.logger.debug("Published WebApplicationContext of servlet '" + getServletName() +
                    "' as ServletContext attribute with name [" + attrName + "]");
        }
    }

    return wac;
}
//容器刷新完毕后接收到事件后初始化9大组件
protected void finishRefresh() {
    // Initialize lifecycle processor for this context.
    initLifecycleProcessor();
```

```
        // Propagate refresh to lifecycle processor first.
        getLifecycleProcessor().onRefresh();

        // Publish the final event.
        //发布IOC容器刷新完毕事件
        publishEvent(new ContextRefreshedEvent(this));

        // Participate in LiveBeansView MBean, if active.
        LiveBeansView.registerApplicationContext(this);
    }
    //收到事件调用onRefresh()方法初始化springMvc的9大组件
    public void onApplicationEvent(ContextRefreshedEvent event) {
        this.refreshEventReceived = true;
        onRefresh(event.getApplicationContext());
    }
    protected void onRefresh(ApplicationContext context) {
        //初始化9大组件
        initStrategies(context);
    }
```

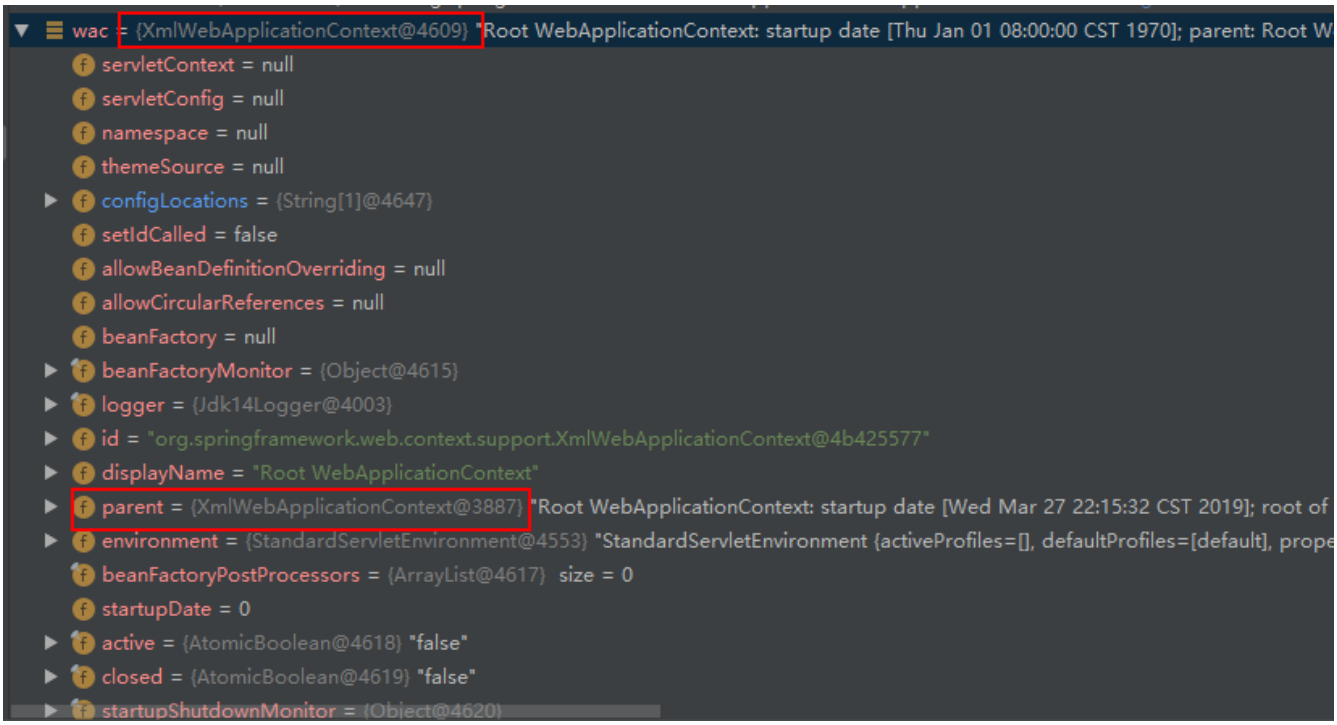

图2:子容器

情况3.使用springboot默认配置好的springMVC模块

1.springboot应用启动后创建容器并刷新容器

```
SpringApplic.java文件
public ConfigurableApplicationContext run(String... args) {
    StopWatch stopWatch = new StopWatch();
    stopWatch.start();
    ConfigurableApplicationContext context = null;
    FailureAnalyzers analyzers = null;
    configureHeadlessProperty();
    SpringApplicationRunListeners listeners = getRunListeners(args);
```

```
    listeners.starting();
    try {
        ApplicationArguments applicationArguments = new DefaultApplicationArguments(
                args);
        ConfigurableEnvironment environment = prepareEnvironment(listeners,
                applicationArguments);
        Banner printedBanner = printBanner(environment);
        context = createApplicationContext();
        analyzers = new FailureAnalyzers(context);
        prepareContext(context, environment, listeners, applicationArguments,
                printedBanner);
        //刷新容器
        refreshContext(context);
        afterRefresh(context, applicationArguments);
        listeners.finished(context, null);
        stopWatch.stop();
        if (this.logStartupInfo) {
            new StartupInfoLogger(this.mainApplicationClass)
                    .logStarted(getApplicationLog(), stopWatch);
        }
        return context;
    }
    catch (Throwable ex) {
        handleRunFailure(context, listeners, analyzers, ex);
        throw new IllegalStateException(ex);
    }
}
```

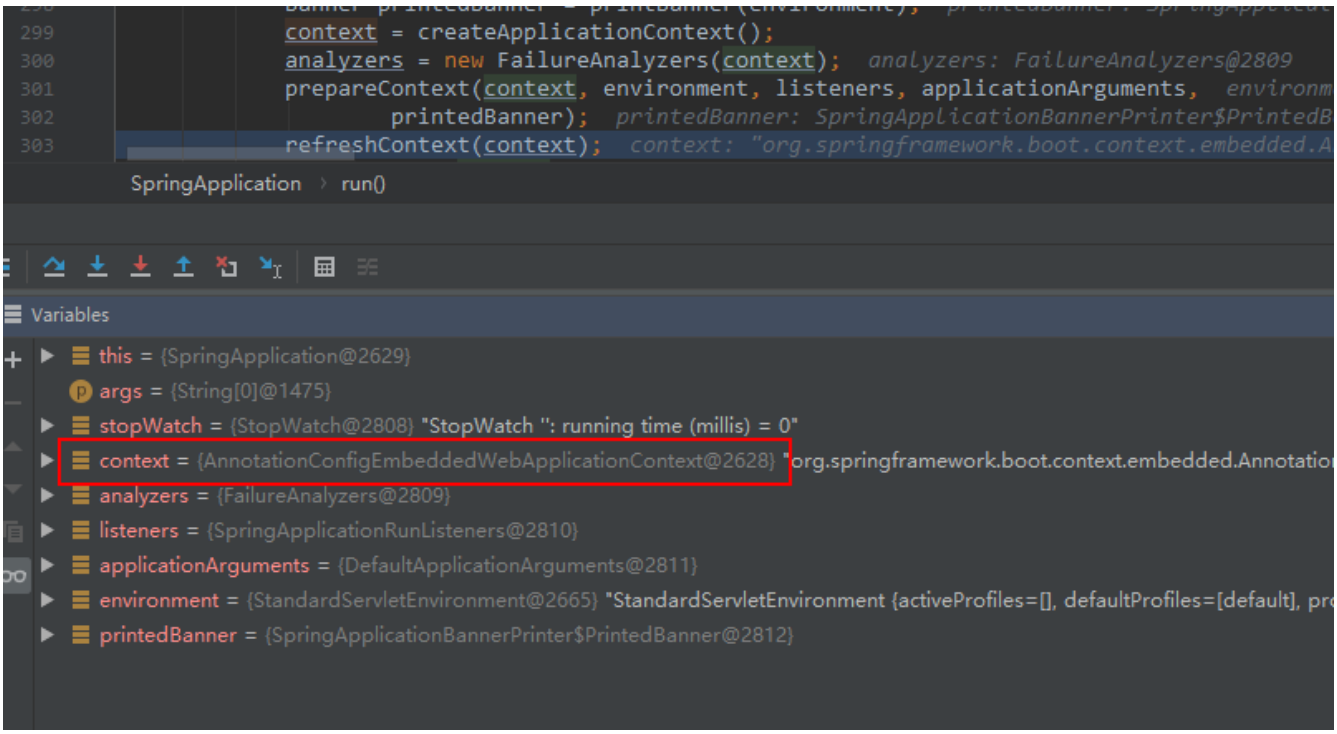

图3:springboot应用启动时创建的容器

2.当发送请求访问资源时检查是否存在IOC容器有就直接拿来用并初始化9大组件,下一次请求时不再做IOC容器是否存在的检查

```
当发送请求服务器收到请求后,检校IOC容器如果有直接拿来使用
@Override
```

```
protected final void initServletBean() throws ServletException {
    getServletContext().log("Initializing Spring FrameworkServlet '" + getServletName() + "'");
    if (this.logger.isInfoEnabled()) {
        this.logger.info("FrameworkServlet '" + getServletName() + "': initialization started");
    }
    long startTime = System.currentTimeMillis();

    try {
        this.webApplicationContext = initWebApplicationContext();
        initFrameworkServlet();
    }
    catch (ServletException ex) {
        this.logger.error("Context initialization failed", ex);
        throw ex;
    }
    catch (RuntimeException ex) {
        this.logger.error("Context initialization failed", ex);
        throw ex;
    }

    if (this.logger.isInfoEnabled()) {
        long elapsedTime = System.currentTimeMillis() - startTime;
        this.logger.info("FrameworkServlet '" + getServletName() + "': initialization completed in
" +
                elapsedTime + " ms");
    }
}
protected WebApplicationContext initWebApplicationContext() {
    //拿到IOC容器
    WebApplicationContext rootContext =
            WebApplicationContextUtils.getWebApplicationContext(getServletContext());
    WebApplicationContext wac = null;

    if (this.webApplicationContext != null) {
        // A context instance was injected at construction time -> use it
        wac = this.webApplicationContext;
        if (wac instanceof ConfigurableWebApplicationContext) {
            ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) wac;
            if (!cwac.isActive()) {
                // The context has not yet been refreshed -> provide services such as
                // setting the parent context, setting the application context id, etc
                if (cwac.getParent() == null) {
                    // The context instance was injected without an explicit parent -> set
                    // the root application context (if any; may be null) as the parent
                    cwac.setParent(rootContext);
                }
                //
                configureAndRefreshWebApplicationContext(cwac);
            }
        }
    }
    if (wac == null) {
        // No context instance was injected at construction time -> see if one
        // has been registered in the servlet context. If one exists, it is assumed
        // that the parent context (if any) has already been set and that the
        // user has performed any initialization such as setting the context id
        wac = findWebApplicationContext();
```

```
    }
    if (wac == null) {
        // No context instance is defined for this servlet -> create a local one
        wac = createWebApplicationContext(rootContext);
    }

    if (!this.refreshEventReceived) {
        // Either the context is not a ConfigurableApplicationContext with refresh
        // support or the context injected at construction time had already been
        // refreshed -> trigger initial onRefresh manually here.
        //直接调用onRefresh()创建9大组件
        onRefresh(wac);
    }

    if (this.publishContext) {
        // Publish the context as a servlet context attribute.
        String attrName = getServletContextAttributeName();
        getServletContext().setAttribute(attrName, wac);
        if (this.logger.isDebugEnabled()) {
            this.logger.debug("Published WebApplicationContext of servlet '" + getServletName() +
                    "' as ServletContext attribute with name [" + attrName + "]");
        }
    }

    return wac;
}
//创建9大组件
protected void onRefresh(ApplicationContext context) {
    initStrategies(context);
}
```

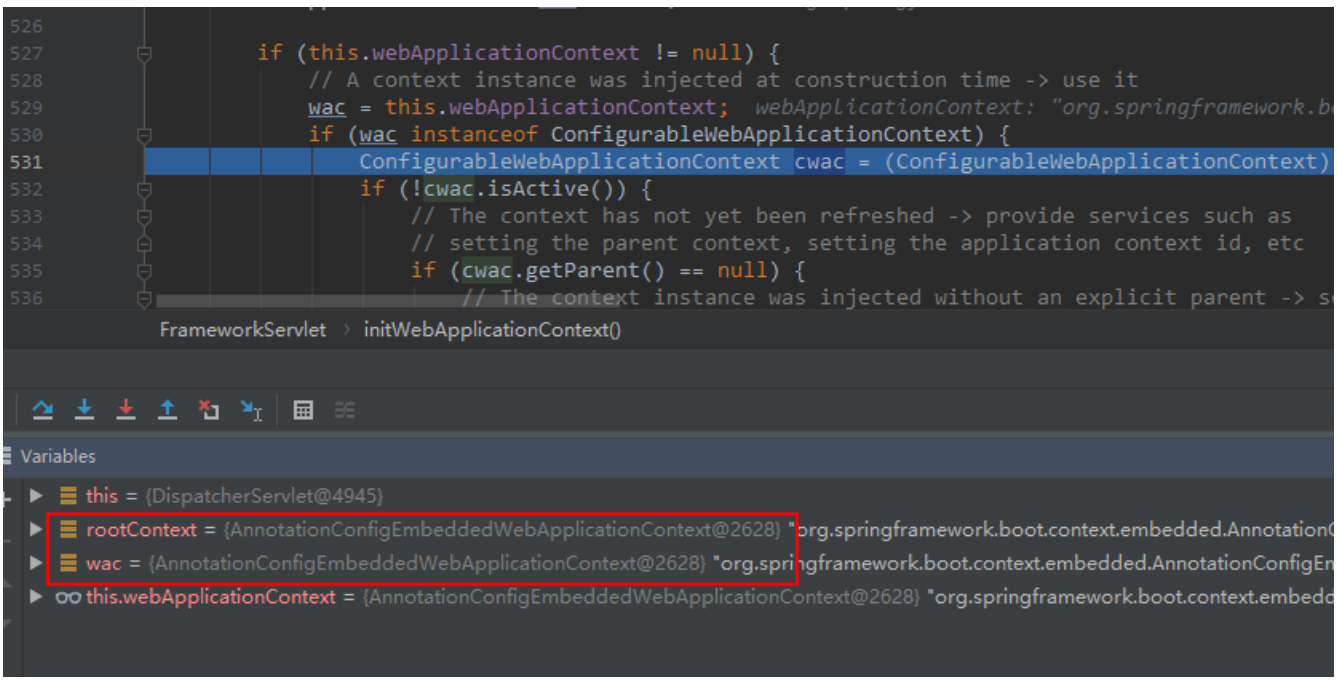

图4:再次拿到springboot应用启动时创建的容器

总结:

情况1下是服务器启东时就创建IOC容器并初始化9大组件

情况2下是服务器启东时创建父IOC容器以及子IOC容器并初始化9大组件，其实是有些不同点的后面篇章会说到

情况3下是springboot应用启动时就创建一个IOC容器,待第一次请求来时拿到已经创建好的IOC容器并初始化9大组件，其实这很好理解是为什么,因为springboot本来就是模块的，当需要什么模块时就直接插入什么模块的starter即可,那么springMVC作为一个模块，只需要在应用启动时创建一个IOC容器即可，后续的springMVC自己的组件只需要在后面初始化好就行,这很符合springboot的设计原则。